



AMD Graphics Developer Performance Tools

Jonathan Zarge
Seth Sowerby
AMD Graphics - Performance Tools

March 07, 2007

Overview

- Graphics Performance Tools Overview
- GPU PerfStudio
- GPU ShaderAnalyzer
- Tootle
- Other AMD Graphics Tools
- Summary

Graphics Performance Tools Overview

AMD Graphics Tools

- Real-time performance analysis
 - GPU PerfStudio – Interactive graphics tuning
- Static resource analysis
 - GPU ShaderAnalyzer - Shader performance analysis
 - Tootle - Triangle mesh optimization

External Graphics Tools

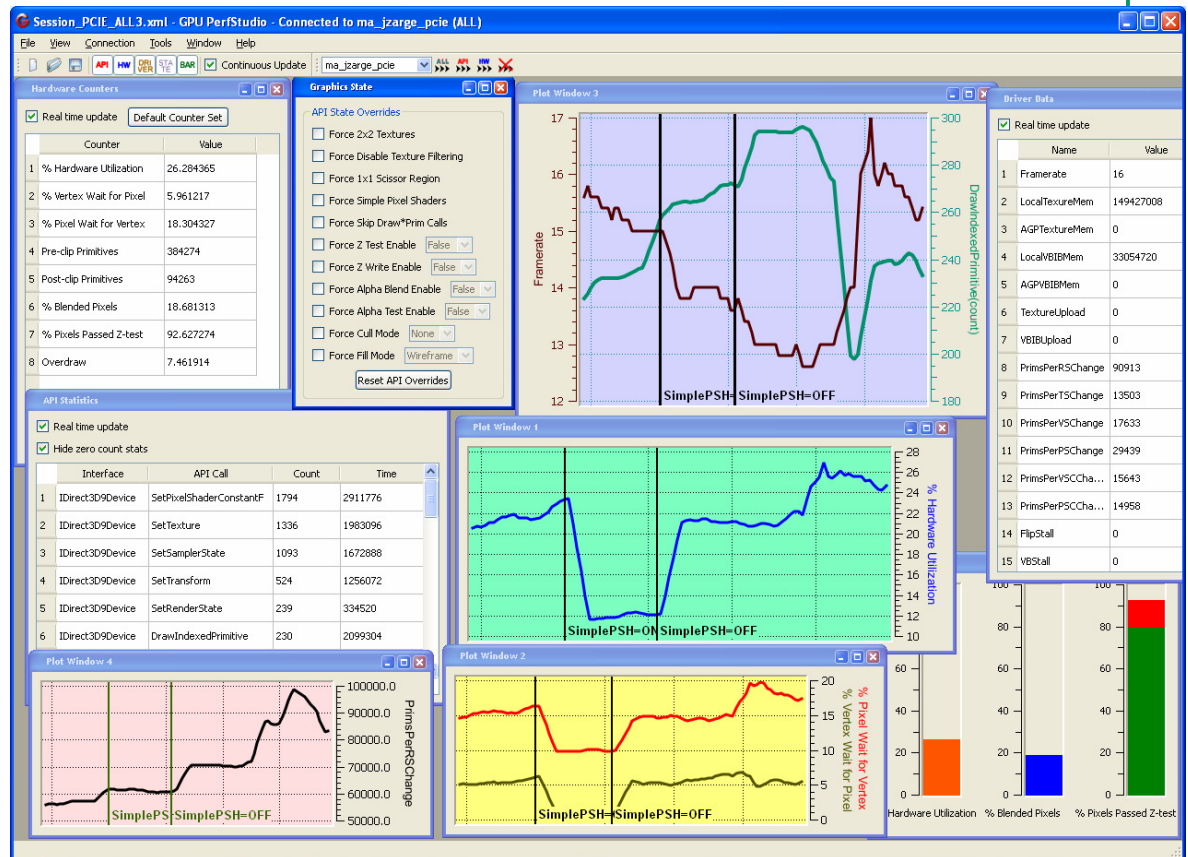
- PIX plugin – support for AMD counters (DirectX)
- gDEDebugger - support for AMD counters (OpenGL)

GPU PerfStudio

GPU PerfStudio Overview

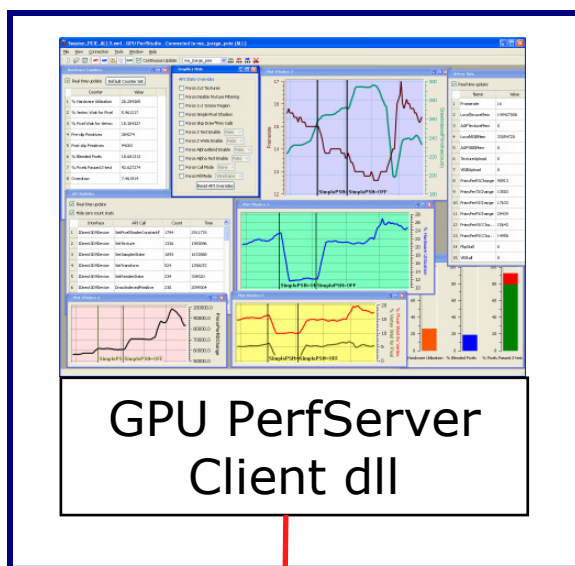


- Monitor performance while D3D or OpenGL application executes remotely
- Real-time visualization:
 - API statistics
 - Hardware/driver data
- Override rendering states
- Launch remote applications
- Flexible data plotting

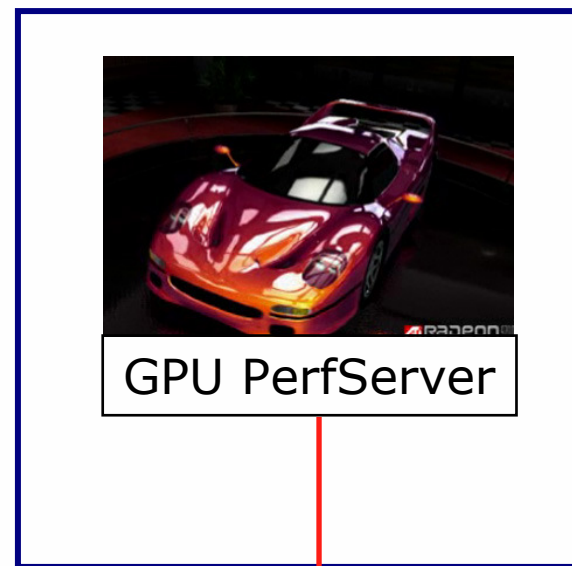


GPU PerfStudio: Client/Server

Client Machine



Server Machine



TCP/IP

← Data
Commands →

GPU PerfStudio: New Features

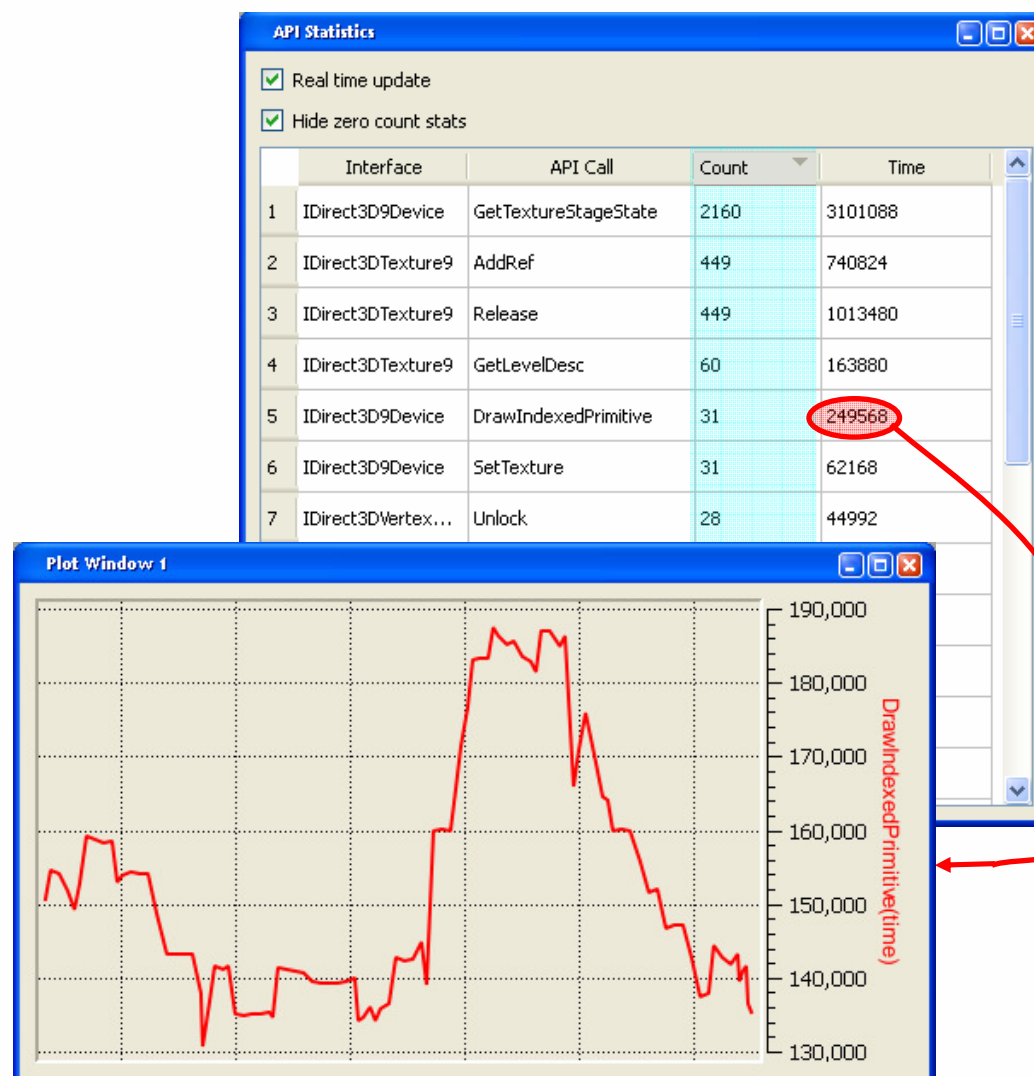
- DX10/Vista/R600
 - ability to analyze cutting edge applications on the newest hardware
- OpenGL
 - support for OpenGL applications
- Render target and non-RT state overrides
 - Especially useful for image-space effects
 - depth of field, glows, render to texture
- Data filters
 - For gaining additional insight into your data
 - average, median, min, max, derivative
- Plot marker lines
 - Demonstrate the effect of state override on performance data

GPU PerfStudio: New Features (cont.)

- Flexible bar charts
 - Control colors, sizes, alarm, range
- Table cell formatting
 - Customize the layout of your tables
 - Cell size, color, font
- Convenience features
 - Application remembers recent sessions, recent server machines, recent applications, etc.
- Selectable antialiasing
 - Fine tune the look and performance of GPU PerfStudio

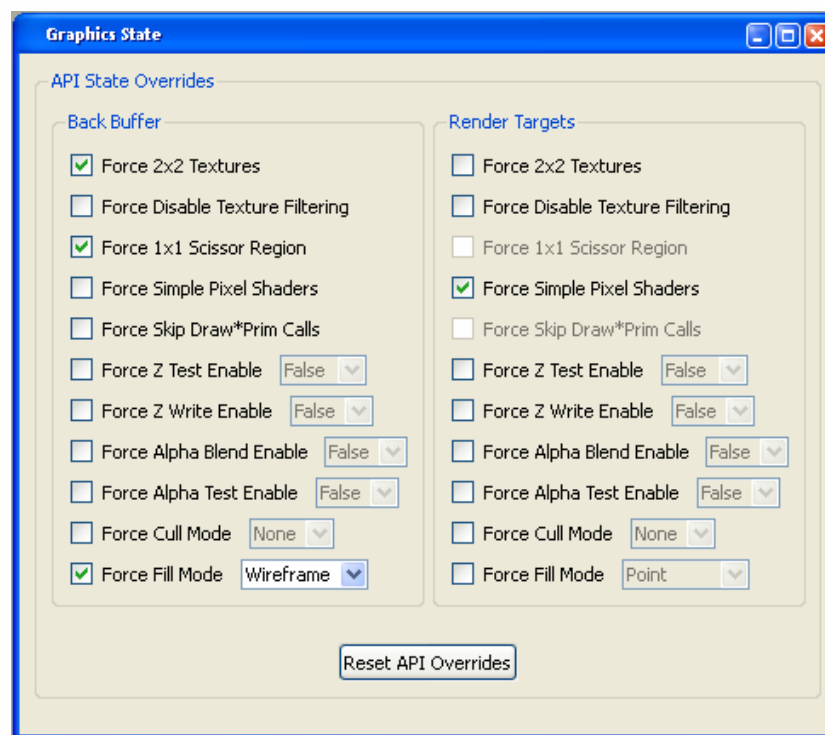
GPU PerfStudio Features: API Statistics

- Per-frame D3D and OpenGL API call data
- Sorting of API call counts and times
- Flexible plotting of all numeric data
- Plot window properties control appearance



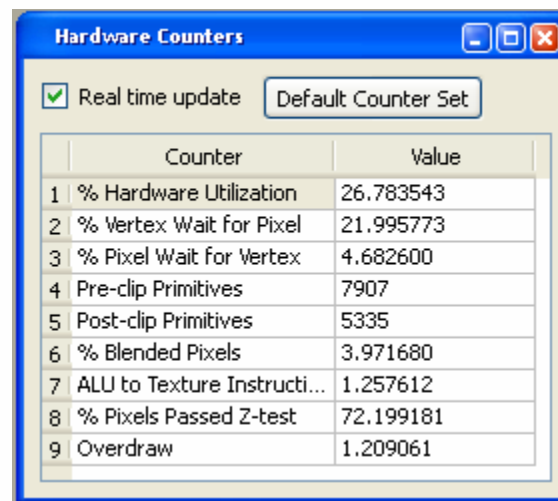
GPU PerfStudio Features: API Statistics

- Real-time state overrides
 - Render Target and non-RT state overrides
 - Useful for full-screen effects



GPU PerfStudio Features: Hardware

- Hardware counters include:
 - % Hardware Utilization
 - Primitive counts
 - Pixel vs. vertex bounded
 - % Pixels passed Z-test
- Editable table properties
 - Cell size, font, colors
- Plotting of all numeric data
 - Filters include: average, median, min, max, derivative

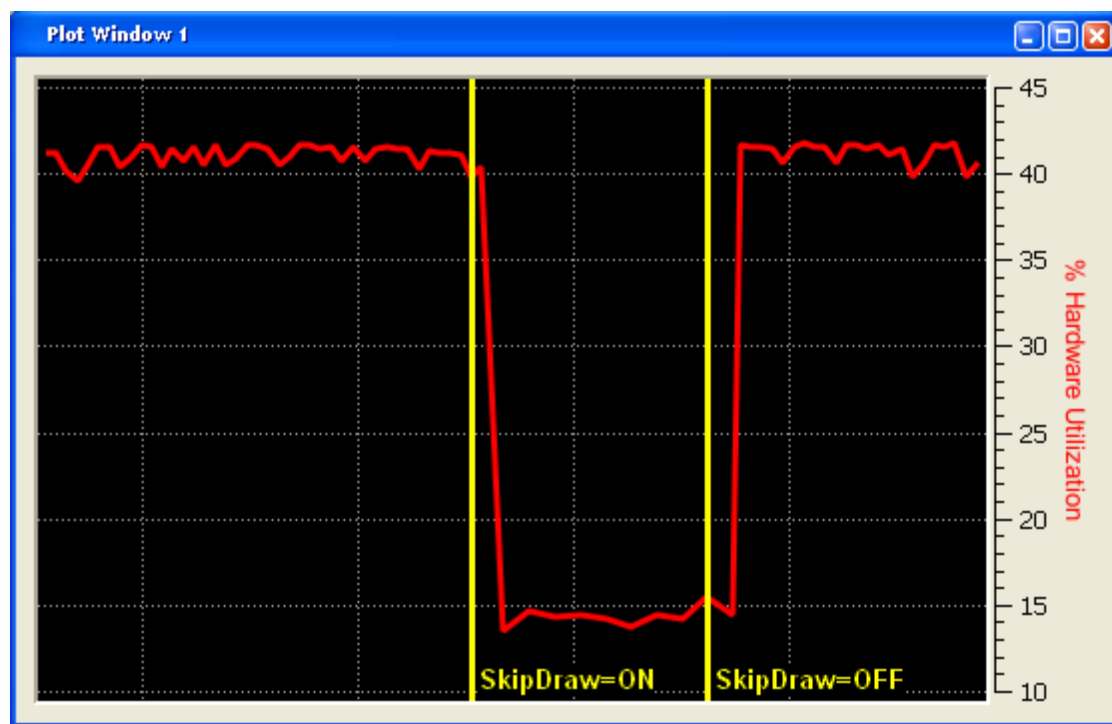


	Counter	Value
1	% Hardware Utilization	26.783543
2	% Vertex Wait for Pixel	21.995773
3	% Pixel Wait for Vertex	4.682600
4	Pre-clip Primitives	7907
5	Post-clip Primitives	5335
6	% Blended Pixels	3.971680
7	ALU to Texture Instructi...	1.257612
8	% Pixels Passed Z-test	72.199181
9	Overdraw	1.209061



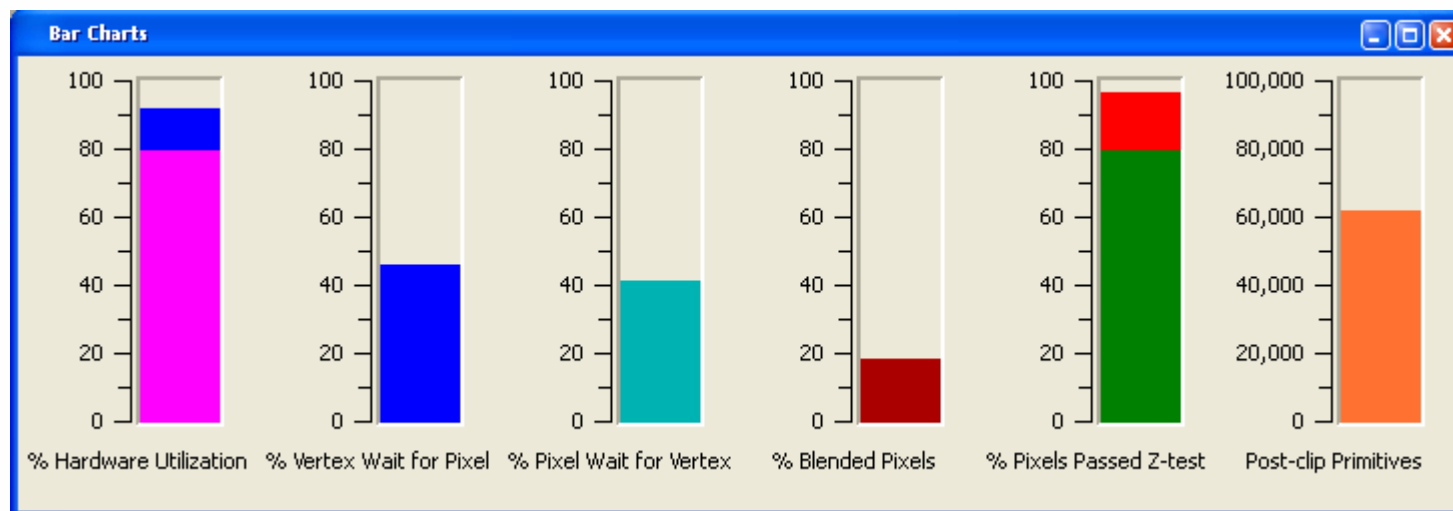
GPU PerfStudio Features: Plotting

- Marker lines to indicate frame of API state change
- Multiple data series on single plot
- Plot properties:
 - Data axis
 - Axis range
 - Grid axis
 - Tick label format
 - Line width
 - Marker properties
 - Filter properties
 - Antialiasing
 - Plot colors



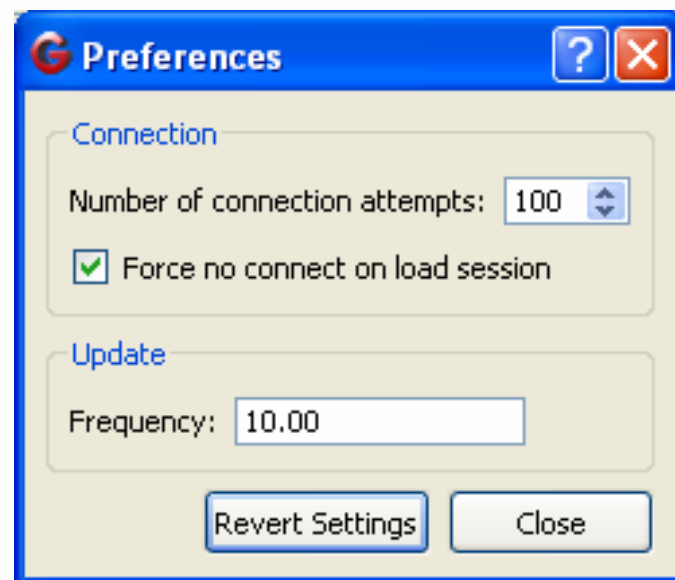
GPU PerfStudio Features: Bar Charts

- Flexible bar charting mechanism
- Customizable “alarm” level
- Tick label format
- Minimum and maximum range values
- Data filtering



GPU PerfStudio Preferences

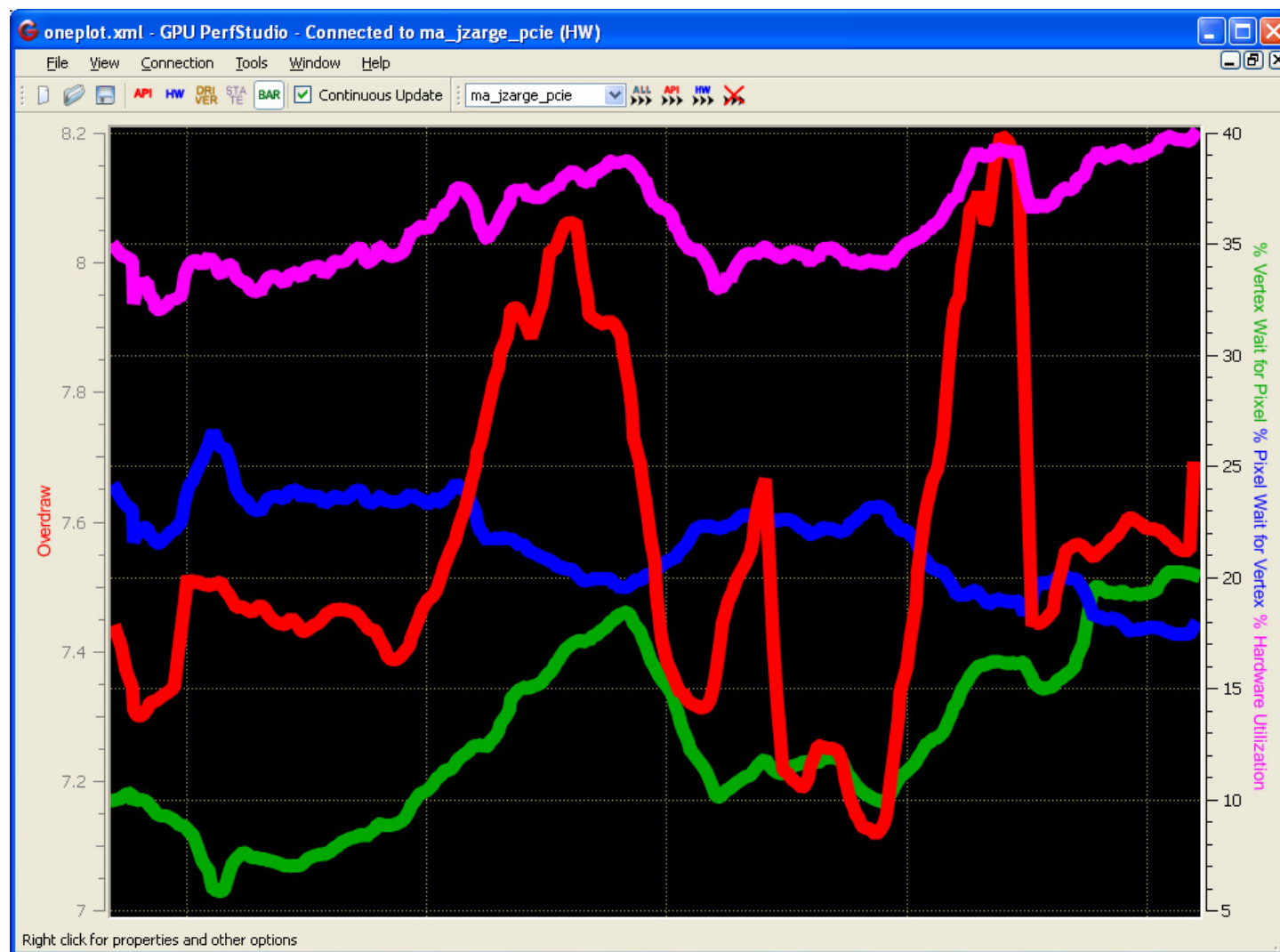
- Preferences are application based, not per session
- Connection preferences:
 - Number of connection attempts
 - Force GPU PerfStudio to not connect when a session is opened
- Modify the update frequency - number of updates per second



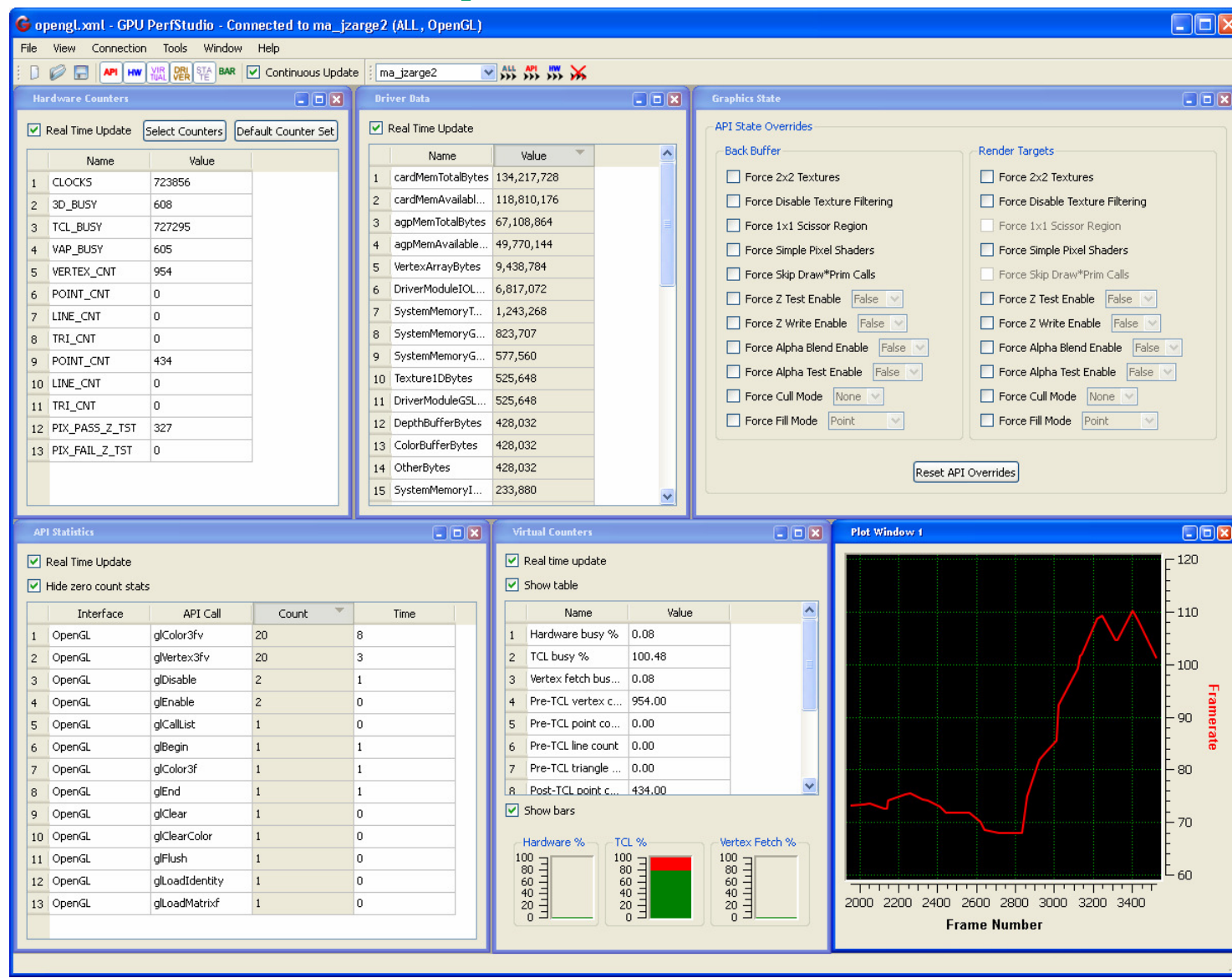
Performance Tuning w/GPU PerfStudio

- CPU vs GPU balance
 - Is the GPU saturated?
- Pixel/Vertex waiting
 - Is vertex processing the bottleneck?
- Pixel shader bottleneck
- Draw*Primitive per state change
 - Good indication of batch size
- Inefficient use of API
- High memory usage
- Texture bandwidth and filtering
- Efficient use of z-buffer and sorting
- Wireframe for geometry visualization

GPU PerfStudio Demo



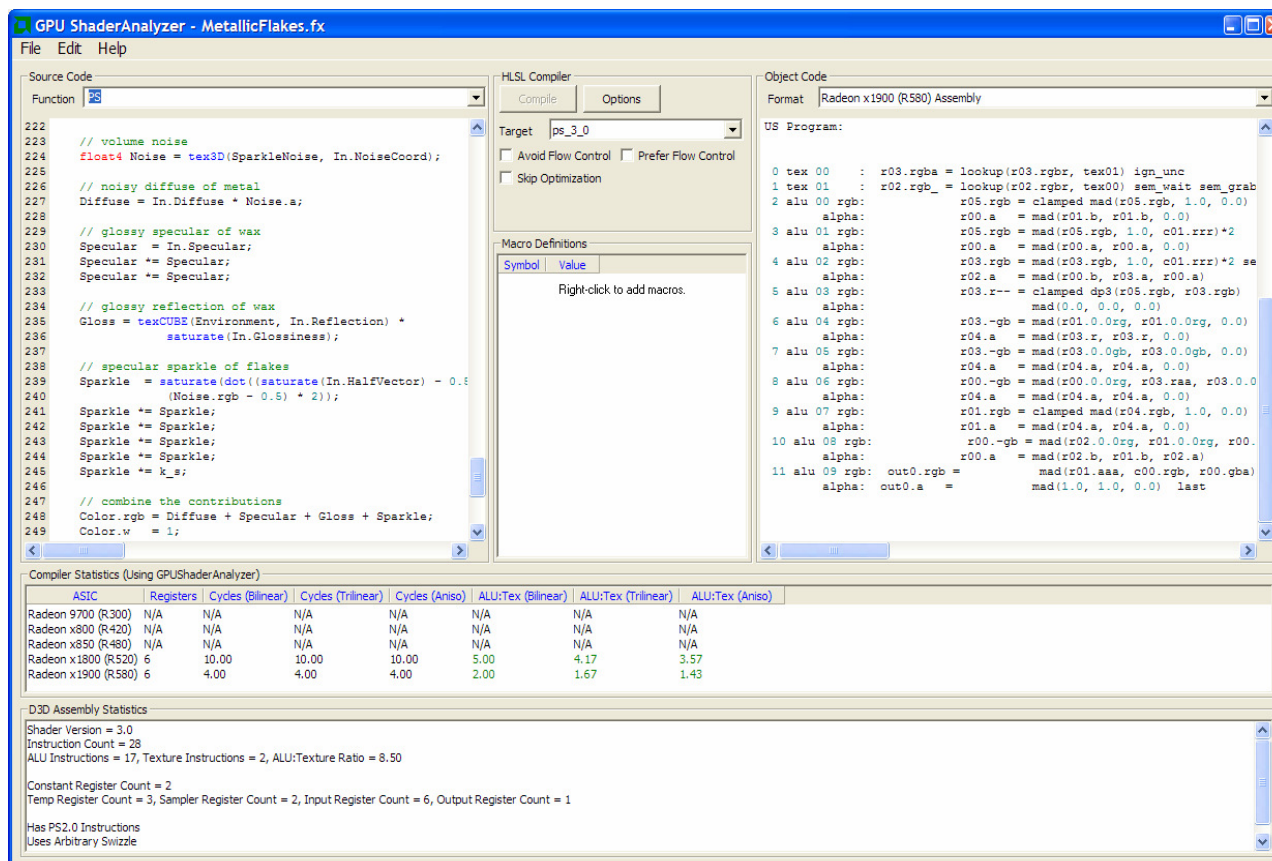
GPU PerfStudio - OpenGL



GPU ShaderAnalyzer

GPU ShaderAnalyzer

- Shader performance analysis tool
- Shader tuning environment
- Instant performance feedback as you tune your shaders



Source Code

```

222 // volume noise
223 float4 Noise = tex3D(SparkleNoise, In.NoiseCoord);
224
225 // noisy diffuse of metal
226 Diffuse = In.Diffuse * Noise.a;
227
228 // glossy specular of wax
229 Specular = In.Specular;
230 Specular *= Specular;
231 Specular *= Specular;
232 Specular *= Specular;
233
234 // glossy reflection of wax
235 Gloss = texCUBE(Environment, In.Reflection) *
236 saturate(In.Glossiness);
237
238 // specular sparkle of flakes
239 Sparkle = saturate(dot((saturate(In.HalfVector) - 0.5),
240 (Noise.rgb - 0.5) * 2));
241 Sparkle *= Sparkle;
242 Sparkle *= Sparkle;
243 Sparkle *= Sparkle;
244 Sparkle *= Sparkle;
245 Sparkle *= k_s;
246
247 // combine the contributions
248 Color.rgb = Diffuse + Specular + Gloss + Sparkle;
249 Color.w = 1;

```

HLSL Compiler

Target: ps_3_0

☐ Avoid Flow Control ☐ Prefer Flow Control

☐ Skip Optimization

Macro Definitions

Right-click to add macros.

Object Code

Format: Radeon x1900 (R580) Assembly

US Program:

```

0 tex 00 : r03.rgba = lookup(r03.rgb, tex01) ign_unc
1 tex 01 : r02.rgb_ = lookup(r02.rgb, tex00) sem_wait sem_grab
2 alu 00 rgb: r05.rgb = clamped mad(r05.rgb, 1.0, 0.0)
alpha: r00.a = mad(r01.b, r01.b, 0.0)
3 alu 01 rgb: r05.rgb = mad(r05.rgb, 1.0, c01.rzz)*2
alpha: r00.a = mad(r00.a, r00.a, 0.0)
4 alu 02 rgb: r03.rgb = mad(r03.rgb, 1.0, c01.rzz)*2 se
alpha: r02.a = mad(r00.b, r03.a, r00.a)
5 alu 03 rgb: r03.r-- = clamped dp3(r05.rgb, r03.rgb)
alpha: mad(0.0, 0.0, 0.0)
6 alu 04 rgb: r03.-gb = mad(r01.0.rg, r01.0.0rg, 0.0)
alpha: r04.a = mad(r03.r, r03.r, 0.0)
7 alu 05 rgb: r03.-gb = mad(r03.0.0gb, r03.0.0gb, 0.0)
alpha: r04.a = mad(r04.a, r04.a, 0.0)
8 alu 06 rgb: r00.-gb = mad(r00.0.0rg, r03.raa, r03.0.0)
alpha: r04.a = mad(r04.a, r04.a, 0.0)
9 alu 07 rgb: r01.rgb = clamped mad(r04.rgb, 1.0, 0.0)
alpha: r01.a = mad(r04.a, r04.a, 0.0)
10 alu 08 rgb: r00.-gb = mad(r02.0.0rg, r01.0.0rg, r00.
alpha: r00.a = mad(r02.b, r01.b, r02.a)
11 alu 09 rgb: out0.rgb = mad(r01.aaa, c00.rgb, r00.gba)
alpha: out0.a = mad(1.0, 1.0, 0.0) last

```

Compiler Statistics (Using GPUShaderAnalyzer)

ASIC	Registers	Cycles (Bilinear)	Cycles (Trilinear)	Cycles (Aniso)	ALU:Tex (Bilinear)	ALU:Tex (Trilinear)	ALU:Tex (Aniso)
Radeon 9700 (R300)	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Radeon x800 (R420)	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Radeon x850 (R480)	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Radeon x1800 (R520)	6	10.00	10.00	10.00	5.00	4.17	3.57
Radeon x1900 (R580)	6	4.00	4.00	4.00	2.00	1.67	1.43

D3D Assembly Statistics

Shader Version = 3.0
Instruction Count = 28
ALU Instructions = 17, Texture Instructions = 2, ALU:Texture Ratio = 8.50

Constant Register Count = 2
Temp Register Count = 3, Sampler Register Count = 2, Input Register Count = 6, Output Register Count = 1

Has PS2.0 Instructions
Uses Arbitrary Swizzle

GSA Features: Performance Analysis

Compiler Statistics (Using Catalyst 7.1)							
ASIC	Registers	Cycles (Bilinear)	Cycles (Trilinear)	Cycles (Aniso)	ALU:Tex (Bilinear)	ALU:Tex (Trilinear)	ALU:Tex (Aniso)
Radeon 9700 (R300)	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Radeon x800 (R420)	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Radeon x850 (R480)	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Radeon x1800 (R520)	4	8.00	8.00	8.00	2.00	1.67	1.43
Radeon x1900 (R580)	4	4.00	4.80	5.60	1.00	0.83	0.71

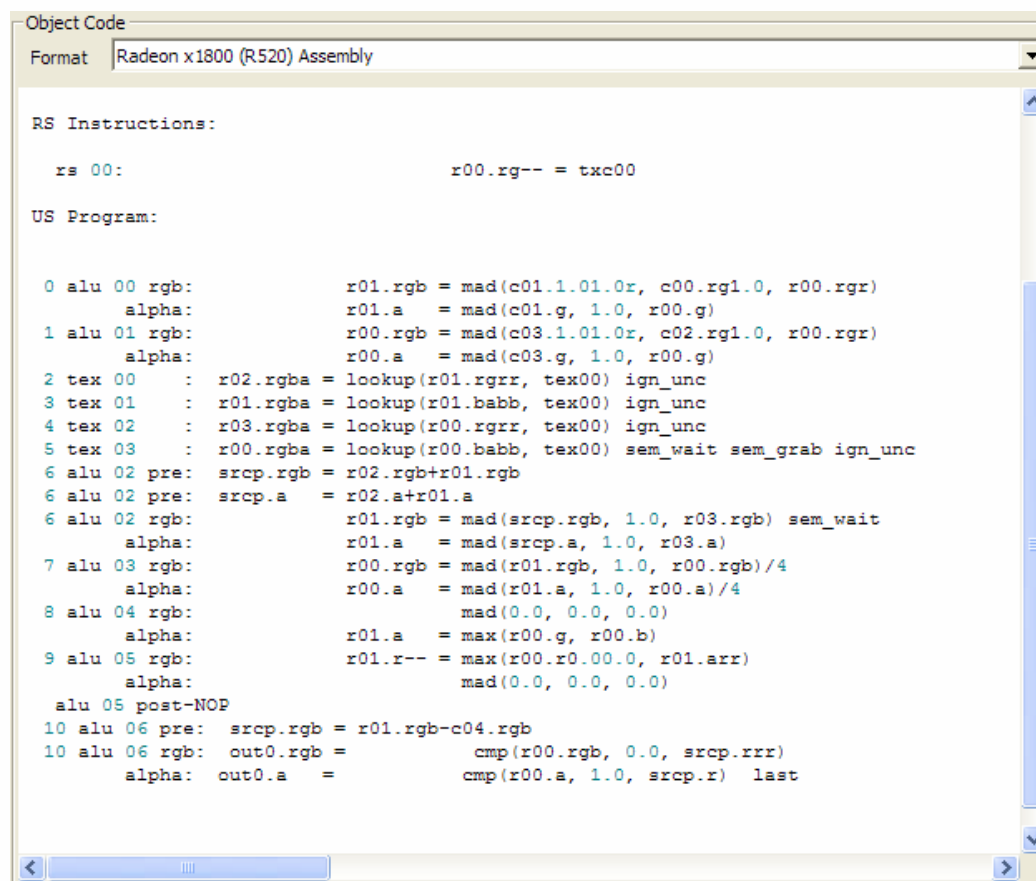
- Predicts shader performance on range of AMD GPUs.
- Analyzes the compiled hardware instruction stream.
- Displays estimated cycle count & ALU:Texture instruction ratio.
- Color codes ALU:Tex ratio based in whether shader is ALU or texture bound.
- Analysis is tied to specific Catalyst driver releases.

GSA Features: Performance Analysis

- Analysis method considers static & dynamic flow control.
- Calculates the cost of each side of a branch to calculate the minimum, maximum & average number of cycles to execute.
- Factors in the expected flow control coherence.
- Currently only average cycles is displayed but minimum & maximum available from command line analysis.
- Estimates cost of texture instructions with bilinear, trilinear & anisotropic filtering.
- Cost based on typical texture fetch cost, not theoretical maximum.

GSA Features: Hardware Disassembly

- View the actual shader as executed by the hardware.
- Shows the effects of optimizations made by the hardware shader compiler.
- No wondering where your shader performance is going.
- Can also display D3D shader disassembly.



```

Object Code
Format: Radeon x1800 (R520) Assembly

RS Instructions:

rs 00:                                r00.rg-- = txc00

US Program:

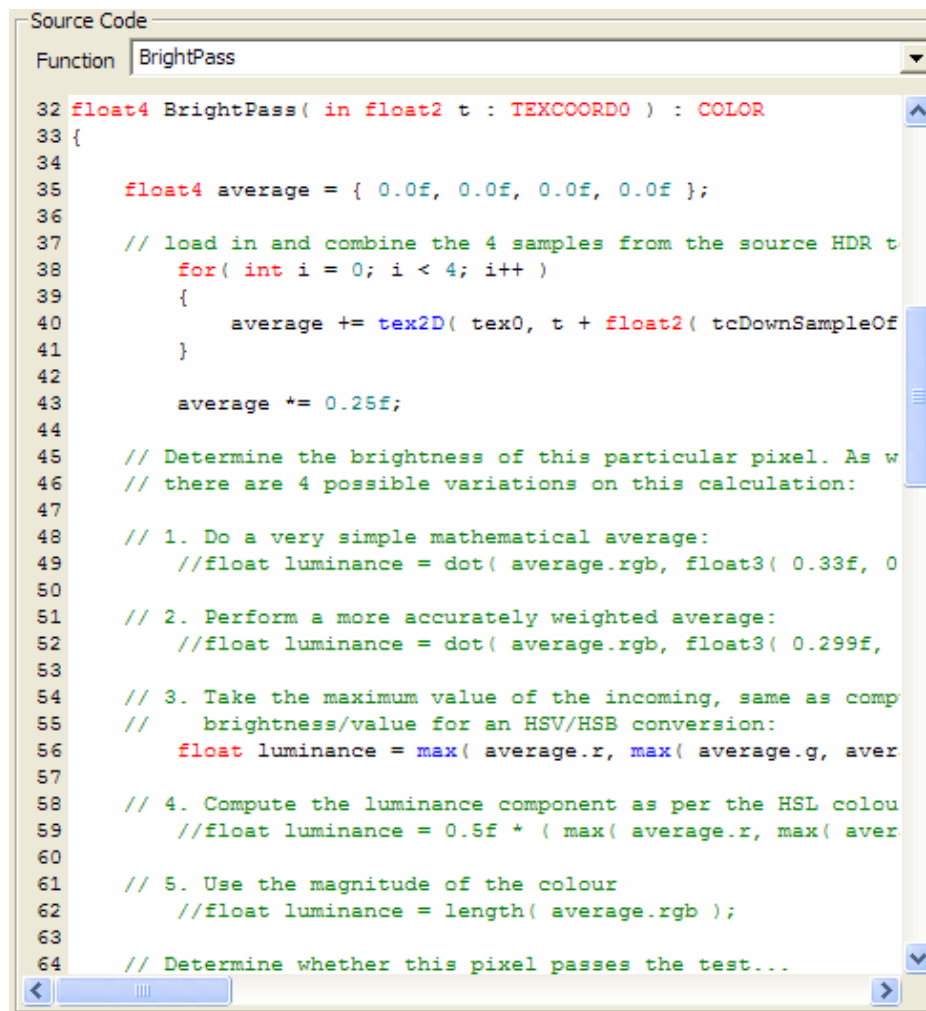
0 alu 00 rgb:                          r01.rgb = mad(c01.1.01.0r, c00.rg1.0, r00.rgr)
  alpha:                               r01.a = mad(c01.g, 1.0, r00.g)
1 alu 01 rgb:                          r00.rgb = mad(c03.1.01.0r, c02.rg1.0, r00.rgr)
  alpha:                               r00.a = mad(c03.g, 1.0, r00.g)
2 tex 00 : r02.rgba = lookup(r01.rgr, tex00) ign_unc
3 tex 01 : r01.rgba = lookup(r01.babb, tex00) ign_unc
4 tex 02 : r03.rgba = lookup(r00.rgr, tex00) ign_unc
5 tex 03 : r00.rgba = lookup(r00.babb, tex00) sem_wait sem_grab ign_unc
6 alu 02 pre: srcp.rgb = r02.rgb+r01.rgb
6 alu 02 pre: srcp.a = r02.a+r01.a
6 alu 02 rgb:                          r01.rgb = mad(srcp.rgb, 1.0, r03.rgb) sem_wait
  alpha:                               r01.a = mad(srcp.a, 1.0, r03.a)
7 alu 03 rgb:                          r00.rgb = mad(r01.rgb, 1.0, r00.rgb)/4
  alpha:                               r00.a = mad(r01.a, 1.0, r00.a)/4
8 alu 04 rgb:                          mad(0.0, 0.0, 0.0)
  alpha:                               r01.a = max(r00.g, r00.b)
9 alu 05 rgb:                          r01.r-- = max(r00.r0.00.0, r01.arr)
  alpha:                               mad(0.0, 0.0, 0.0)
alu 05 post-NOP
10 alu 06 pre: srcp.rgb = r01.rgb-c04.rgb
10 alu 06 rgb: out0.rgb =               cmp(r00.rgb, 0.0, srcp.rrr)
  alpha: out0.a =                       cmp(r00.a, 1.0, srcp.r) last
  
```

GSA – Supported Shader Formats

- DX9 Pixel\Vertex Shaders
- DX 1.1 – DX3.0 Assembly Shaders
- DX 2.0 – DX3.0 HLSL Shaders
- DX10 Pixel\Vertex\Geometry Shaders
- GLSL Pixel\Vertex Shaders
- arb_fp\arb_vp programs

GSA Features: Shader Editing

- Fully featured editor.
- All the aids you have come to rely on.
 - Cut\Copy\Paste.
 - Undo\Redo.
 - Etc.
- Context coloring.



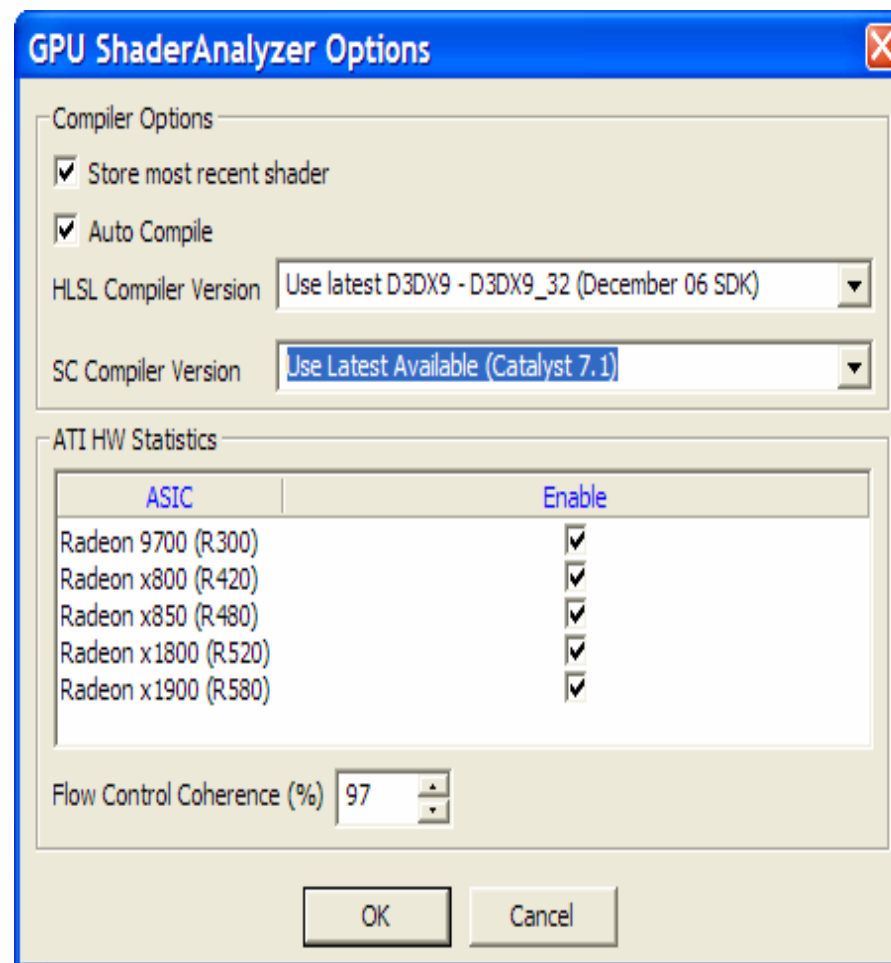
```

Source Code
Function: BrightPass

32 float4 BrightPass( in float2 t : TEXCOORD0 ) : COLOR
33 {
34
35     float4 average = { 0.0f, 0.0f, 0.0f, 0.0f };
36
37     // load in and combine the 4 samples from the source HDR t
38     for( int i = 0; i < 4; i++ )
39     {
40         average += tex2D( tex0, t + float2( tcDownSampleOf
41     }
42
43     average *= 0.25f;
44
45     // Determine the brightness of this particular pixel. As w
46     // there are 4 possible variations on this calculation:
47
48     // 1. Do a very simple mathematical average:
49     //float luminance = dot( average.rgb, float3( 0.33f, 0
50
51     // 2. Perform a more accurately weighted average:
52     //float luminance = dot( average.rgb, float3( 0.299f,
53
54     // 3. Take the maximum value of the incoming, same as comp
55     // brightness/value for an HSV/HSB conversion:
56     float luminance = max( average.r, max( average.g, aver
57
58     // 4. Compute the luminance component as per the HSL colou
59     //float luminance = 0.5f * ( max( average.r, max( aver
60
61     // 5. Use the magnitude of the colour
62     //float luminance = length( average.rgb );
63
64     // Determine whether this pixel passes the test...
    
```


GSA Features: Options

- Configure which HLSL compiler to use.
- Currently supports Microsoft D3DX compilers from Feb 05-current.
- Configure which version of the ATI Shader Compiler to use.
- Track performance against driver releases.
- Configure which GPUs to analyze performance for.



GSA Features: Command Line Support

- Shader performance analysis & hardware disassembly also available from command line.
- Analyze a single shader or a search a directory tree for shaders.
- Output analysis to .csv file for further analysis within MS Excel.

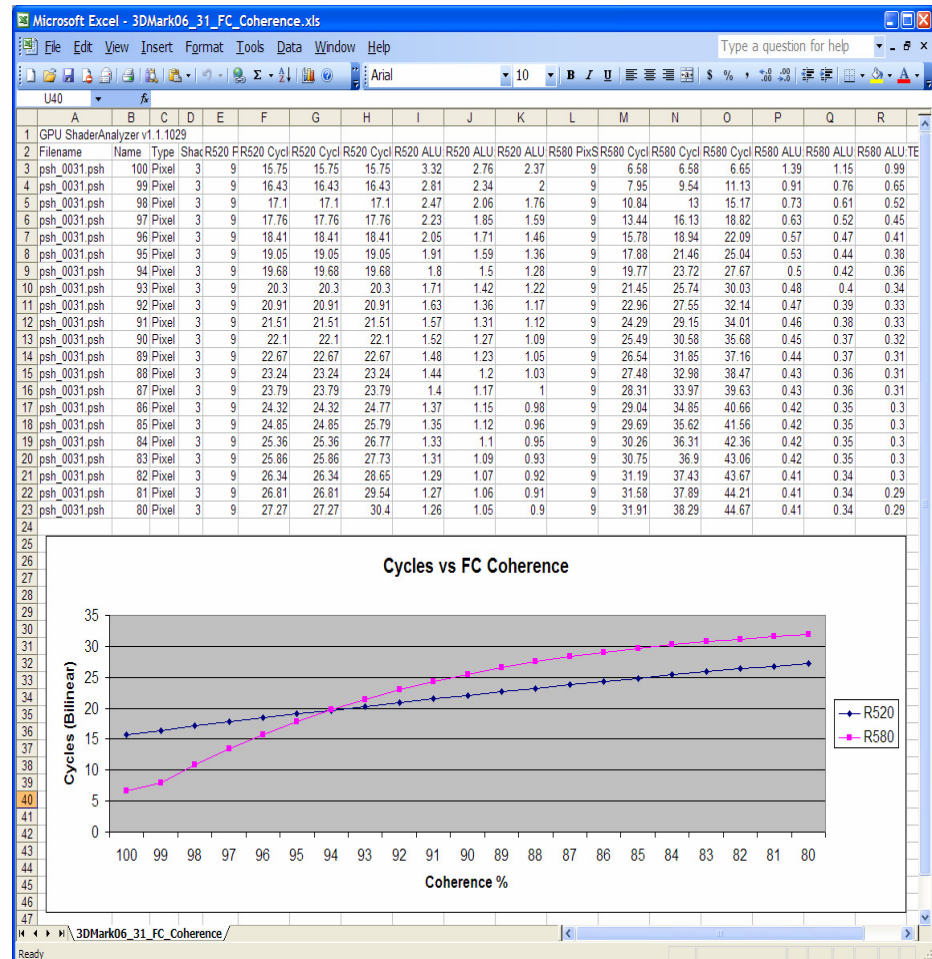
GSA Features: CSV Analysis Output



CSV output & MS Excel
gives endless options for
deeper investigation.

Example:

- Pixel shader with flow control run through GSA with different FC coherence values set.
- Graph performance against coherence.



GSA: Demo

The screenshot displays the GPU ShaderAnalyzer (GSA) interface for the file 'MetallicFlakes.fx'. The interface is divided into several panes:

- Source Code:** Contains the HLSL source code for the 'PS' function, including volume noise, noisy diffuse, glossy specular, and sparkle calculations.
- HLSL Compiler:** Includes a 'Compile' button, 'Options' button, a 'Target' dropdown set to 'ps_3_0', and checkboxes for 'Avoid Flow Control', 'Prefer Flow Control', and 'Skip Optimization'.
- Macro Definitions:** A table with 'Symbol' and 'Value' columns, currently empty with a 'Right-click to add macros' instruction.
- Object Code:** Shows the generated assembly code for the 'US Program' in 'Radeon x1900 (R580) Assembly' format, listing texture lookups, register assignments, and arithmetic operations.
- Compiler Statistics (Using GPUShaderAnalyzer):** A table comparing various AMD GPUs across different metrics.
- D3D Assembly Statistics:** Provides detailed statistics about the shader, including version, instruction counts, and register usage.

Compiler Statistics (Using GPUShaderAnalyzer)

	ASIC	Registers	Cycles (Bilinear)	Cycles (Trilinear)	Cycles (Aniso)	ALU:Tex (Bilinear)	ALU:Tex (Trilinear)	ALU:Tex (Aniso)
Radeon 9700 (R300)	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Radeon x800 (R420)	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Radeon x850 (R480)	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Radeon x1800 (R520)	6	10.00	10.00	10.00	5.00	4.17	3.57	
Radeon x1900 (R580)	6	4.00	4.00	4.00	2.00	1.67	1.43	

D3D Assembly Statistics

Shader Version = 3.0
 Instruction Count = 28
 ALU Instructions = 17, Texture Instructions = 2, ALU:Texture Ratio = 8.50

Constant Register Count = 2
 Temp Register Count = 3, Sampler Register Count = 2, Input Register Count = 6, Output Register Count = 1

Has PS2.0 Instructions
 Uses Arbitrary Swizzle

Tootle

Tootle: Overview

- A Triangle Order Optimization Tool.
- Provided as a library for integration into your tool-chain.
- Simple to use.

Tootle: What It Does

- Improves vertex-cache hit rate
 - Shade fewer vertices
- Reduces overdraw
 - Shade fewer pixels
 - View independent
- Win-win scenario

Tootle: Background

- Based on I3D 2006 paper by Nehab\Barczak\Sander.
- Uses D3DXOptimizeMesh to perform vertex cache optimization.
- Uses D3D for overdraw measurement.

Tootle: Overdraw Reduction

- Example Scene
 - 70k polygons
 - 10 materials
- Reduced overdraw by factor of two.
- 3-7% performance increase compared to D3DXOptimizeMesh.



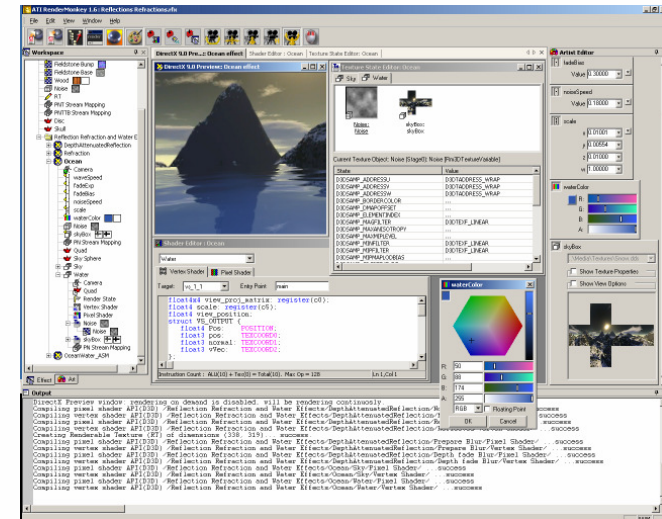
Tootle: Why you should use it

- It's free !
 - No runtime cost.
- It's free !!
 - Optimizes both vertex & pixel pipelines.
- It's free !!!
 - Works on all hardware.
- It's free !!!!
 - To use.

AMD Content Creation Tools

AMD Content Creation Tools

- RenderMonkey
 - Shader development environment
 - Supports HLSL, D3D assembler and GLSL
 - New version coming soon
 - OpenGL ES 2.0 support
 - GPU ShaderAnalyzer integration
- The Compressorator
 - Tool for compressing textures
 - Creates mip-map levels
- CubeMapGen
 - Tool for creating filtered cube maps without seams
 - Uses angular extent filtering
- NormalMapper
 - Automatic normal map generation tool



Other AMD Sessions

- 10:30-11:30 Massive Multi-Core CPUs and Gaming
 - Justin Boggs
- 12:00-13:00 Performance Profiling With AMD GPU Tools: A Case Study Abstract
 - Holger Gruen
- 12:00-13:00 OpenGL ES 2.0: Start Developing Now (Room 3009)
 - Dan Ginsburg
- 13:30-14:30 Cool GLSL Tips (Khronos Theater)
 - Bill Licea-Kane
- 14:30-15:30 Frostbite Rendering Architecture and Real-Time Procedural Shading and Texturing Techniques
 - Natalya Tatarchuk (with Johan Andersson from DICE)
- 16:00-17:00 Procedural Tools and Techniques for Current and Future Game Platforms
 - Sebastien Deguy (Allegorithmic) & Jeremy Shopf (AMD)
- 9:00-10:00 Massive Multi-Core CPUs and Gaming (Tomorrow, Room 2010)
 - Justin Boggs

Wrap Up

- AMD GPU Performance Tools are there to make you job easier.
- DX9 versions available now.
- DX10 & OpenGL support coming soon.
- Grab them from ati.amd.com/developer/
- Send us feedback – we want to hear the good & the bad.

Questions?

Jonathan.Zarge@amd.com

Seth.Sowerby@amd.com

GPUTools.Support@amd.com

<http://ati.amd.com/developer/>

Trademark Attribution

AMD, the AMD Arrow logo and combinations thereof are trademarks of Advanced Micro Devices, Inc. in the United States and/or other jurisdictions. Other names used in this presentation are for identification purposes only and may be trademarks of their respective owners.

©2006 Advanced Micro Devices, Inc. All rights reserved.